Globally Solving Nonconvex QPs via Completely Positive Programming*

Jieqiu Chen[†] Samuel Burer[‡]

February 3, 2011

Abstract

Nonconvex quadratic programming is an NP-hard problem that optimizes a general quadratic function over linear constraints. This paper introduces a new global optimization algorithm for this problem, which combines two ideas from the literature—finite branching based on the first-order KKT conditions and polyhedral-semidefinite relaxations of completely positive (or copositive) programs. Through a series of computational experiments comparing the new algorithm with existing codes on a diverse set of test instances, we demonstrate that the new algorithm is an attractive method for globally solving nonconvex QP.

Keywords: nonconvex quadratic programming, global optimization, branch-and-bound, semidefinite programming, copositive programming, completely positive programming.

1 Introduction

We consider the problem of optimizing a general quadratic function subject to linear and bound constraints:

$$\begin{aligned} \min \quad & \frac{1}{2}x^T H x + f^T x && \text{(QP)} \\ \text{s.t.} \quad & Ax \le b \\ & A_{eq}\, x = b_{eq} \\ & l \le x \le u \end{aligned}$$

where $x \in \Re^n$ is the variable and $H \in \Re^{n\times n}$, $f \in \Re^n$, $A \in \Re^{m\times n}$, $b \in \Re^m$, $A_{eq} \in \Re^{m_{eq}\times n}$, $b_{eq} \in \Re^{m_{eq}}$, $l \in \Re^n$, and $u \in \Re^n$ are the data. Without loss of generality, H is symmetric, and we assume H is *not* positive semidefinite, which implies generally that (QP) is nonconvex and NP-hard (Pardalos and Vavasis, 1991). Components of the vectors l and u are allowed to be infinite, but we assume the feasible set of (QP) is bounded. We also assume without loss of generality that (QP) has an interior point and A_{eq} has full row rank.

Problem (QP) arises in many application areas such as inventory theory (Lootsma and Pearson, 1970), scheduling (Skutella, 2001), and free boundary problems (Lin and Cryer, 1985). Moré

*The research of both authors was supported in part by NSF Grant CCF-0545514. Jieqiu Chen was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

[†]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. Email: jieqchen@mcs.anl.gov

[‡]Department of Management Sciences, University of Iowa, Iowa City, IA, 52242-1994, USA. Email: samuel-burer@uiowa.edu

and Toraldo (1989) mention several applications of the box-constrained case, i.e., when the only constraints are $l \le x \le u$, and make a connection with the linear complementarity problem. Extensions of (QP) that incorporate additional nonlinearities—for example, variants of the trust-region subproblem within trust-region methods (Celis et al., 1985) and subproblems in robust linear regression (Nguyen and Welsch, 2010)—are relevant in practice. In this sense, (QP) is also an important substructure occurring in other optimization problems.

Because (QP) is nonconvex, a local minimum may not be a global minimum. Many methods for (QP) employ nonlinear programming techniques such as active-set or interior-point methods to calculate critical points that satisfy the Karush-Kuhn-Tucker (KKT) conditions with a good objective value. Gould and Toint (2002) survey numerous such methods, and the function `quadprog` in Matlab's Optimization Toolbox (MathWorks, 2010) is a widely available implementation. A variety of techniques are also available to find a global solution of (QP), and Pardalos (1991) surveys several of these global optimization methods. A recent method, which is closely related to this paper, is by Burer and Vandenbussche (2008). Excellent off-the-shelf software packages, such as BARON (Sahinidis, 1996) and Couenne (Belotti, 2010), are also available for the solution of (QP).

In this paper, we propose a new method for globally optimizing (QP). Our approach follows that of Burer and Vandenbussche (2008) by employing a finite branch-and-bound (B&B) scheme, in which branching is based on the first-order KKT conditions and polyhedral-semidefinite relaxations are solved at each node of the B&B tree. However, our method differs from theirs in that our relaxations have a different structure and are solved with a different algorithm, which exploits this structure.

In particular, our relaxations are derived from the notions of *completely positive* (CP) and *doubly nonnegative* (DNN) programs as specified by Burer (2009, 2010). (Completely positive programs are also known as *copositive programs*.) We connect (QP) with CP programs by reformulating (QP) as a quadratic program with linear equality, nonnegativity, and complementarity constraints. Burer (2009) shows that such problems can be formulated as CP programs, which are convex programs that optimize a linear function over the convex cone of completely positive matrices subject to linear constraints. However, since even CP programs are NP-hard, one must relax them in practice. Relaxing a CP program in a natural way yields a DNN program, whose distinguishing feature is its matrix variable, which is both positive semidefinite and entry-wise nonnegative. Furthermore, Burer (2010) develops a specialized algorithm for these DNN programs and demonstrates that similar quality bounds can be obtained in less time compared to the type of relaxations employed by Burer and Vandenbussche (2008).

The main goal of this paper is to demonstrate that merging finite KKT-based B&B with completely positive programming yields an attractive method for globally solving (QP). To show this, we conduct computational experiments on 243 diverse instances of (QP) and compare with the method of Burer and Vandenbussche (2008) and with the general-purpose global solver Couenne. The results demonstrate that our algorithm solves (QP) faster than the one by Burer and Vandenbussche (2008) on most, but not all, of the test instances. In addition, our algorithm performs well compared to Couenne. For example, on nearly all instances which took Couenne more than 1,000 seconds to solve, our algorithm requires about the same or less time. In addition, Couenne runs out of time or memory more often than our algorithm does on the test instances.

A secondary goal of this paper is to provide a convenient global optimization solver for (QP) that can be used by practitioners and researchers alike. Our implementation is based in Matlab, uses the same syntax as the local optimization routine `quadprog`, and requires only an external linear programming solver. In contrast, the method by Burer and Vandenbussche (2008) requires a convex quadratic programming solver. In Section 5, we illustrate the convenience of our implementation by using it as a subroutine to solve a more general nonlinear program motivated by trust-region methods.

The paper is organized as follows. In Section 2, we provide background on the finite KKT-based B&B method of Burer and Vandenbussche (2008) and the approach of Burer (2009, 2010) involving CP and DNN programs. In Section 3, we describe our reformulation of (QP) as a CP program using the KKT conditions and discuss the important technical issue of bounding the dual variables in the CP formulation. Section 4 combines the ingredients from Sections 2–3 to describe our B&B implementation and conduct the computational results. Section 5 details the trust-region example, and finally in Section 6, we conclude the paper with a few brief remarks.

1.1 Notation and terminology

We let $e \in \Re^n$ represent the vector of all ones. For an index i and vector x, x_i denotes the i-th entry of x. For an index set $F \subseteq \{1, \dots, n\}$, $x_F \in \Re^{|F|}$ is defined as the vector composed of entries of x that are indexed by F. $X \succeq 0$ means that the matrix X is symmetric positive semidefinite, and $\mathrm{diag}(X)$ denotes the main diagonal of X as a vector. The inner product of two matrices $A, B \in \Re^{n\times n}$ is defined as $A \bullet B := \mathrm{trace}(A^T B)$; $\circ$ represents the Hadamard product, i.e., component-wise product.

2 Background

In this section, we briefly review the finite B&B method (Burer and Vandenbussche, 2008) and the CP and DNN techniques (Burer, 2009, 2010) mentioned in the Introduction.

2.1 The finite branch-and-bound method

The finite B&B method proposed in Burer and Vandenbussche (2008) works by first incorporating into (QP) its first-order KKT conditions and then enforcing the first-order KKT conditions through branching. The primary advantage of such an approach is that the B&B tree is finite, unlike other global optimization approaches based on spatial branching. A secondary advantage is that this approach allows one to develop stronger relaxations of (QP), e.g., ones that involve the KKT conditions.

For the sake of simplicity, we discuss finite branching with respect to a simplified version of (QP) that has no equality constraints and no explicit lower and upper bounds, i.e., the feasible set is simply $\{x : Ax \le b\}$.

Specifically, as illustrated by the following program, the authors start with the quadratic program (1)–(2) and incorporate its first-order KKT conditions (3)–(4):

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x && (1)\\
\text{s.t.} \quad & Ax \le b && (2)\\
& A^T\gamma = Hx + f, \quad \gamma \ge 0 && (3)\\
& (b - Ax)_i \gamma_i = 0 \quad \forall\, i = 1, \dots, m. && (4)
\end{aligned}
$$

Then, at each node in the B&B tree, the authors introduce index sets $F^{(b-Ax)}, F^{\gamma} \subseteq \{1, \dots, m\}$ and replace (4) by

$$
\begin{aligned}
(b - Ax)_i = 0 \quad & \forall\, i \in F^{(b-Ax)}\\
\gamma_i = 0 \quad & \forall\, i \in F^{\gamma}.
\end{aligned} \qquad (5)
$$

Note that (5) enforces the complementarity conditions $(b - Ax)_i\gamma_i = 0$ only for those $i \in F^{(b-Ax)} \cup F^\gamma$. At the root node, the authors set $F^{(b-Ax)} = F^\gamma = \emptyset$, and then the idea is to branch by adding indices to $F^{(b-Ax)}$ or F^γ, progressively enforcing more complementarities at nodes deeper in the tree. In particular, branching on a node involves selecting an index $i \in \{1,\dots,m\} \backslash (F^{(b-Ax)} \cup F^y)$ and creating two children by adding i to $F^{(b-Ax)}$ for one child and adding i to F^y for the other. Branching maintains $F^{(b-Ax)} \cap F^y = \emptyset$ always, and a leaf node is characterized by $F^{(b-Ax)} \cup F^y = \{1,\dots,m\}$, which enforces (4) fully.

At any node, note that (1)–(3) and (5) still constitute a nonconvex problem due to the quadratic objective. Thus, a convex relaxation is constructed and solved to compute a lower bound for that node. There are many choices for the convex relaxations, and a particular SDP relaxation is constructed by Burer and Vandenbussche (2008). With that SDP relaxation, the authors show that any leaf node can be pruned and thus their B&B scheme is correct and finite. Their proof relies primarily on proposition 2.1 of Burer and Vandenbussche (2008) and the boundedness of the SDP relaxation. We mention that, for our method to be presented in Section 3, the same types of results can be shown ensuring the finiteness and correctness of our algorithm also.

2.2 Doubly nonnegative programs

Consider a quadratic program having linear equality, nonnegativity, and complementarity constraints:

$$\begin{aligned} \min \quad & \frac{1}{2}\tilde{x}^T \tilde{H} \tilde{x} + \tilde{f}^T \tilde{x} && \text{(NQP)} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = \tilde{b}, \ \tilde{x} \geq 0 \\ & \tilde{x}_i \tilde{x}_j = 0 \quad \forall\ (i,j) \in E && (6) \end{aligned}$$

where E is a fixed set of pairs. Burer (2009) has shown (NQP) is equivalent to a so-called *completely positive program*, which naturally has the following SDP relaxation:

$$\begin{aligned} \min \quad & \frac{1}{2}\tilde{H} \bullet \tilde{X} + \tilde{f}^T \tilde{x} && \text{(DNP)} \\ \text{s.t.} \quad & \tilde{A}\tilde{x} = \tilde{b}, \ \operatorname{diag}(\tilde{A}\tilde{X}\tilde{A}^T) = \tilde{b} \circ \tilde{b} \\ & \tilde{X}_{ij} = 0 \ \ \forall\ (i,j) \in E \\ & \begin{pmatrix} 1 & \tilde{x}^T \\ \tilde{x} & \tilde{X} \end{pmatrix} \succeq 0, \ (\tilde{x}, \tilde{X}) \geq 0, \end{aligned}$$

where the semidefinite matrix is called *doubly nonnegative* because both its eigenvalues and entries are nonnegative. The major contribution by Burer (2010) is an algorithm to solve (DNP) efficiently, yet approximately. The algorithm also produces dual bounds, which makes it appropriate for use in a B&B algorithm. The implementation of the algorithm requires an additional input, namely finite upper bounds on the variable $\tilde{x}$. For example, assuming (NQP) is bounded, such bounds can be gotten in a pre-processing phase by solving several linear programming (LP) problems. Note that these upper bounds may not appear explicitly in (DNP).

Related to this issue of boundedness, in Section 3.2 we discuss how to bound the dual variables arising from the KKT approach of Section 2.1, which will ultimately become components of $\tilde{x}$ in (NQP). A similar issue was faced by Burer and Vandenbussche (2008), and we will discuss our approach for dealing with it. Otherwise, we will not review the details of the algorithm for (DNP) since we can treat it as an available subroutine and simply embed it inside the finite B&B method.

3 Reformulation and Bounding

In this section, we first discuss the steps required to reformulate (QP) as (NQP). In short, (NQP) serves as a kind of standard form, and the reformulation involves techniques such as introducing the KKT system of (QP), representing inequalities as equalities, and shifting and scaling variables. While the transformation is not particularly complicated, it warrants discussion because it is critical to our approach. After the reformulation, we show how to bound the dual variables that arise when formulating the KKT system (see also Section 2.2).

We caution the reader that, in order to simplify the notation, we sometimes abuse it. For example, the same variable x may be used before and after a shifting or scaling.

3.1 Reformulation

There are several ways of reformulating (QP) as (NQP). Our fundamental approach is to incorporate the first-order KKT conditions of (QP) into its formulation. This gives rise to the complementarity conditions in (NQP) that will become the basis of branching as discussed in Section 2.1. This approach also allows us to construct potentially stronger semidefinite relaxations.

Since we employ the finite B&B framework, it is also desirable to keep the number of complementarity conditions minimal in order to have a small B&B tree. In fact, we would like the number of complementarities in (NQP) to be exactly the same as the number naturally occurring in the KKT system of (QP). For example, suppose x_j is a free variable in (QP) with $(l_j, u_j) = (-\infty, +\infty)$, which is nevertheless bounded by assumption. In order to put (QP) in the form of (NQP), one could split x_j into the difference of nonnegative variables before constructing the KKT system, but doing so would introduce two more complementarities compared with first constructing the KKT system and then splitting x_j. There is an even better way to handle such x_j as we describe next.

To describe out explicit transformation—the one that has been implemented exactly in our code—we group the variables x_j of (QP) into categories based on the following index sets:

$$\begin{aligned}
L &:= \{j : -\infty < l_j, u_j = +\infty\} && \text{“lower bounds only”} \\
U &:= \{j : -\infty = l_j, u_j < +\infty\} && \text{“upper bounds only”} \\
B &:= \{j : -\infty < l_j, u_j < +\infty\} && \text{“both lower and upper bounds”} \\
F &:= \{j : -\infty = l_j, u_j = +\infty\} && \text{“free”.}
\end{aligned}$$

Using the transformation $x_j \to u_j - l_j$, variables x_j with $j \in U$ can easily be transformed into variables with $j \in L$. In other words, we may assume $U = \emptyset$. In a similar manner, we may also assume $l_j = 0$ for all $j \in L$ and $(l_j, u_j) = (0, 1)$ for all $j \in B$. Then (QP) becomes

$$\begin{aligned}
\min \quad & \frac{1}{2} x^T H x + f^T x \\
\text{s.t.} \quad & Ax \le b \\
& A_{eq}\, x = b_{eq} \\
& 0 \le x_L,\ 0 \le x_B \le e,\ x_F \text{ free.}
\end{aligned} \tag{QP$'$}$$

To form the KKT conditions, we introduce multipliers $\gamma \ge 0$, y free, $\lambda_L \ge 0$, $\lambda_B \ge 0$, and $\rho_B \ge 0$ for the respective inequality constraints of (QP$'$). This gives the Lagrangian

$$\frac{1}{2} x^T H x + f^T x - \gamma^T (b - Ax) - y^T (b_{eq} - A_{eq} x) - \lambda_L^T x_L - \lambda_B^T x_B - \rho_B^T (e - x_B)$$

with optimality conditions

$$\begin{aligned}
&Hx + f + A^T\gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = 0 \\
&Ax \le b,\ A_{eq}x = b_{eq},\ 0 \le x_L,\ 0 \le x_B \le e,\ x_F \text{ free} \\
&\gamma \ge 0,\ y \text{ free},\ \lambda_L \ge 0,\ \lambda_B \ge 0,\ \rho_B \ge 0 \\
&(b - Ax) \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ (e - x_B) \circ \rho_B = 0 \qquad (7)
\end{aligned}$$

where, in the first equation, the vectors λ_L, λ_B, and ρ_B are understood to have zeros in their non-indices. Note we also have the implied complementarity

$$\lambda_B \circ \rho_B = 0, \tag{8}$$

which is valid because $x_i \ge 0$ and $1 - x_i \ge 0$ for all $i \in B$ cannot be active at the same time. We also enforce (8) when branching in the B&B scheme; see Section 4.1 for details.

Since (NQP) only allows equality constraints, we introduce slack variables (s, w_B) to turn inequalities into equalities and arrive at the following:

$$\begin{aligned}
&Hx + f + A^T\gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = 0 && (9)\\
&Ax + s = b,\ A_{eq}x = b_{eq},\ x_B + w_B = e,\ (x_L, x_B, s, w_B) \ge 0,\ x_F \text{ free} && (10)\\
&y \text{ free},\ (\gamma, \lambda_L, \lambda_B, \rho_B) \ge 0 \\
&s \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ w_B \circ \rho_B = 0,\ \lambda_B \circ \rho_B = 0. && (11)
\end{aligned}$$

This format fits (NQP) except that y and x_F are free. Recall our assumption that the feasible set of (QP) is bounded, and so bounds on x_F can be calculated by solving some LPs; more details are given in Section 3.2. We also postpone the discussion of how to bound y to Section 3.2.

Suppose now that y and x_F have available lower bounds. Again, we can shift the variables such that their lower bounds are zeros, and thus obtain

$$\begin{aligned}
&Hx + f + A^T\gamma + A_{eq}^T y - \lambda_L - \lambda_B + \rho_B = r && (12)\\
&Ax + s = b,\ A_{eq}x = b_{eq},\ x_B + w_B = e,\ (x, s, w_B) \ge 0 \\
&(\gamma, y, \lambda_L, \lambda_B, \rho_B) \ge 0 \\
&s \circ \gamma = 0,\ x_L \circ \lambda_L = 0,\ x_B \circ \lambda_B = 0,\ w_B \circ \rho_B = 0,\ \lambda_B \circ \rho_B = 0
\end{aligned}$$

where r is a constant vector resulting from the shift.

Now we are ready to reformulate (QP′) as (NQP) using (12). We stack all the primal variables (x, s, w_B) and the dual multipliers $(\gamma, y, \lambda_L, \lambda_B, \rho_B)$ together to form the new variable $\tilde{x} \ge 0$. It is easy to formulate appropriate $(\tilde{A}, \tilde{b})$ such that $\tilde{A}\tilde{x} = \tilde{b}$ models all the linear equalities in (12), and E is constructed to encapsulate all the complementarity conditions in (12) via the constraints $\tilde{x}_i\tilde{x}_j = 0$ for all $(i, j) \in E$.

3.2 Finite bounds on dual variables

As mentioned in Section 2.2, the algorithm we have chosen for solving (DNP) requires finite upper bounds on the nonnegative variables $\tilde{x}_j$. This means we need explicit finite lower and upper bounds on all the original primal and dual variables, which can easily be translated into upper bounds on the $\tilde{x}_j$'s.

First, we show how to compute bounds on x through LP-based preprocessing. By assumption, the primal feasible set is bounded even though l and u may have non-finite components. Then, if a lower (upper) bound is needed for x_j, we solve the following LP:

$$\min\ (\max) \quad \{x_i : Ax \le b,\ A_{eq}x = b_{eq},\ l \le x \le u\}.$$

Next we discuss the dual variables and start by proving that they are indeed bounded. To bound y, the KKT conditions alone are not sufficient. We further need

$$x^T Hx + f^T x + b^T \gamma + b_{eq}^T y + e^T \rho_B = 0, \tag{13}$$

which is obtained from (9)–(11). Indeed, multiplying x^T on both sides of (9) yields

$$x^T Hx + x^T f + x^T A^T \gamma + x^T A_{eq}^T y - x_L^T \lambda_L - x_B^T \lambda_B + x_B^T \rho_B = 0,$$

and then (10) and (11) simplify the above equality to (13).

To simplify the proof, we assume both x and its slack s are bounded below by 0 and above by e; indeed, we can first compute bounds for x and s and then shift and scale. Then define

$$\hat{A} := \begin{pmatrix} A & I \\ A_{eq} & 0 \end{pmatrix}, \quad \hat{b} := \begin{pmatrix} b \\ b_{eq} \end{pmatrix}, \quad \hat{x} := \begin{pmatrix} x \\ s \end{pmatrix},$$

so that the feasible set can be stated as

$$P := \{\hat{x} \in \Re^{n+m} : \ \hat{A}\hat{x} = \hat{b},\ 0 \le \hat{x} \le e\}.$$

Also let $\hat{y}$ be the dual multiplier for $\hat{A}\hat{x} = \hat{b}$, and introduce $\hat{\lambda} \ge 0$ and $\hat{\rho} \ge 0$ as the dual multipliers for $0 \le \hat{x} \le e$. Note that the dual multipliers in (12) are simply sub-vectors of $(\hat{y}, \hat{\lambda}, \hat{\rho})$, and so the boundedness of $(\hat{y}, \hat{\lambda}, \hat{\rho})$ will imply that of the dual multipliers.

It is further helpful to restate the first equation of (12) in terms of $(\hat{y}, \hat{\lambda}, \hat{\rho})$ as follows:

$$\hat{H}\hat{x} + \hat{f} + \hat{A}^T \hat{y} - \hat{\lambda} + \hat{\rho} = 0, \tag{14}$$

where $\hat{H} := \begin{pmatrix} H & 0 \\ 0 & 0 \end{pmatrix}$, $\hat{f} := \begin{pmatrix} f \\ 0 \end{pmatrix}$. Similarly, the equivalent version of (13) is

$$\hat{x}^T \hat{H}\hat{x} + \hat{f}^T \hat{x} + \hat{b}^T \hat{y} + e^T \hat{\rho} = 0. \tag{15}$$

Note that (15) is nonlinear, and so we first linearize it by applying the reformulation-linearization (RLT) technique of Sherali and Adams (1999). In particular, we introduce a new matrix variable $\hat{X}$ to replace $\hat{x}\hat{x}^T$, i.e., $\hat{X}_{ij} = \hat{x}_i \hat{x}_j$ component-wise, which allows us to write (15) as

$$\hat{H} \bullet \hat{X} + \hat{f}^T \hat{x} + \hat{b}^T \hat{y} + e^T \hat{\rho} = 0. \tag{16}$$

RLT also produces the following valid inequalities:

$$0 \le \hat{X}_{i,j} \le \min\{\hat{x}_i, \hat{x}_j\}, \quad \forall\ 1 \le i \le j \le n+m \tag{17}$$

$$1 - \hat{x}_i - \hat{x}_j + \hat{X}_{i,j} \ge 0, \quad \forall\ 1 \le i \le j \le n+m, \tag{18}$$

which further tightens the relationship between $\hat{X}$ and $\hat{x}$. In light of (17) and (18), the boundedness of $\hat{x}$ implies that of $\hat{X}$ and the following set is bounded:

$$\hat{P} := \left\{ (\hat{x}, \hat{X}) \ : \ \hat{x} \in P,\ (17), (18) \right\}.$$

In other words, the recession cone $\hat{P}^0$ of $\hat{P}$ is trivial, i.e., $\hat{P}^0 = \{0\}$.

We are then ready to prove that $(\hat{y}, \hat{\lambda}, \hat{\rho})$ is bounded. We prove this by demonstrating that the recession cone of the set defined collectively by the primal, dual variables and their associated valid inequalities is empty. The recession cone of P, (14), and (16)–(18) is as follows:

$$R^0 := \left\{ \begin{array}{c} (\triangle\hat{x}, \triangle\hat{X}) \geq 0 \\ \triangle\hat{y} \text{ free} \\ (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0 \end{array} : \begin{array}{c} (\triangle\hat{x}, \triangle\hat{X}) \in \hat{P}^0 \\ \hat{H}\triangle\hat{x} + A^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ \hat{H} \bullet \triangle\hat{X} + \hat{f}^T\triangle\hat{x} + \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

Since $\hat{P}^0 = \{0\}$, R^0 simplifies to

$$R^0 = \left\{ \begin{array}{c} (\triangle\hat{x}, \triangle\hat{X}) = (0,0) \\ (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0 \\ \triangle\hat{y} \text{ free} \end{array} : \begin{array}{c} A^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

Now define

$$R := \left\{ \begin{array}{c} (\triangle\hat{\lambda}, \triangle\hat{\rho}) \geq 0 \\ \triangle\hat{y} \text{ free} \end{array} : \begin{array}{c} \hat{A}^T\triangle\hat{y} - \triangle\hat{\lambda} + \triangle\hat{\rho} = 0 \\ \hat{b}^T\triangle\hat{y} + e^T\triangle\hat{\rho} = 0 \end{array} \right\}.$$

R is the projection of the recession cone of R^0 onto the variables $(\triangle\hat{y}, \triangle\hat{\lambda}, \triangle\hat{\rho})$. Hence, it suffices to show that $R = \{0\}$. We do so by exploiting the assumptions that (QP) has an interior point and A_{eq} has full row rank, which imply that P contains an interior point and $\hat{A}$ has full row rank.

Proposition 3.1. *$R = \{0\}$, and hence the dual variables $(\hat{y}, \hat{\lambda}, \hat{\rho})$ are bounded.*

Proof. Consider the LP $\max\{0 : \hat{x} \in P\}$ and its dual

$$\min \left\{ \hat{b}^T\Delta\hat{y} + e^T\Delta\hat{\rho} : \begin{array}{c} \hat{A}^T\Delta\hat{y} - \Delta\hat{\lambda} + \Delta\hat{\rho} = 0 \\ (\Delta\hat{\lambda}, \Delta\hat{\rho}) \geq 0, \ \Delta\hat{y} \text{ free} \end{array} \right\}.$$

Strong duality ensures $\hat{b}^T\Delta\hat{y} + e^T\Delta\hat{\rho} = 0$ at optimality, and so the dual optimal solution set is precisely R.

Now let $(\Delta\hat{y}^*, \Delta\hat{\lambda}^*, \Delta\hat{\lambda}^*) \in R$ be any optimal solution of the dual, and let $\hat{x}^0$ be an interior point of P, which is obviously an optimal solution to the primal problem. Complementary slackness implies $(\Delta\hat{\lambda}^*, \Delta\hat{\rho}^*) = (0,0)$ since $0 < \hat{x}^0 < e$. Then the dual linear constraint simplifies to $\hat{A}^T\Delta\hat{y}^* = 0$. Since $\hat{A}$ has full row rank, $\Delta\hat{y}^* = 0$. Hence, $(\Delta\hat{y}^*, \Delta\hat{\lambda}^*, \Delta\hat{\lambda}^*) = (0,0,0)$, which demonstrates $R = \{0\}$. □

Now that we know the dual variables $(\hat{y}, \hat{\lambda}, \hat{\rho})$ are bounded, to actually compute bounds for them, we again solve some LPs. The constraints of the LPs are exactly the valid constraints we derived based on optimality conditions and RLT techniques. For example, to compute the lower and upper bounds for the k-th component of $\hat{y}$, we solve the following pair of LPs:

$$\min / \max \ \{ \ \hat{y}_k \ : \ \hat{x} \in P, \ (14), \ (16)\text{-}(18) \ \}. \tag{19}$$

We compute the upper bounds for $\hat{\lambda}$ and $\hat{\rho}$ via the same linear programs but with modified objective functions. These bounds must be computed during pre-processing and reformulation because they are required as input later by the algorithm that solves the relaxation in the form of (DNP) at each node of the B&B scheme.

4 Computational Results

In this section, we describe our implementation of the finite B&B algorithm to globally solve (QP) and compare its performance with two other methods on a diverse set of test problems.

4.1 Implementation details

The implementation consists of two main components: (i) the reformulation of (QP) into (NQP) described in Section 3.1, including the calculation of bounds; and (ii) the B&B scheme described in Section 2.1.

The reformulation includes a preprocessing phase in which we remove fixed variables and check if the assumptions of the method are satisfied, i.e., if (QP) has an interior point and A_{eq} has full row rank. After preprocessing, we recast (QP) as (QP$'$) by flipping some variables and shifting and scaling others. Then we formulate the first-order KKT conditions of (QP$'$) as detailed in Section 3.1. With the KKT system available, we compute bounds for each primal and dual variable by solving LPs, e.g., ones of the form (19) for the dual variables. These LPs are the most time-consuming part of the reformulation.[1] The last step of the reformulation puts the problem into the form (NQP) and also scales each of the nonnegative variables in (NQP) to have an implicit upper bound of 1, which is done because we found it improved the numerical precision of the overall algorithm. Also, it facilitates our branching rule as described below.

The B&B algorithm, as with all such algorithms, involves four types of calculations: upper bounds, lower bounds, fathoming, and branching. At each node of the tree, we apply a nonlinear programming method to obtain a local solution of (NQP) without the complementarity condition (6), which provides a currently best global upper bound (GUB). In our implementation, we use the `quadprog` function in Matlab's Optimization Toolbox to solve the QP. `Quadprog` finds locally optimal solution to QPs in a fast and reliable way. At any given node, this calculation is initialized with the value of $\tilde{x}$ from the node's relaxation, which increases the diversity of the search for a good GUB throughout the tree. For lower bounds, at each node we solve the DNN relaxation (DNP) of (NQP) tailored only to respect the complementarities enforced at that node. For example, we solve the root relaxation with no complementarity conditions enforced. At any particular node, if the lower bound obtained by solving the relaxation is within a pre-specified tolerance of GUB, then that node is fathomed. If not, then branching involves the enforcement of the complementarity conditions by setting variables to 0 as discussed in Section 2.1. Specifically, a violated complementarity condition, $\tilde{x}_i\tilde{x}_j > 0$, is selected and $\tilde{x}_i\tilde{x}_j = 0$ is enforced in the two children nodes, where one node enforces $\tilde{x}_i = 0$ and the other node enforces $\tilde{x}_j = 0$.

More details of the B&B algorithm are as follows:

- As mentioned in Section 2.2, we solve the relaxations (DNP) via the specialized decomposition technique proposed in Burer (2010), which is practically efficient and always provides a valid lower bound on the original QP. In particular, the relaxation would be practically expensive to solve by interior-point methods.
- When selecting the next active node to solve, we choose the one that has the lowest lower bound.

[1] We also experimented with bounding the sum of several nonnegative variables at a time (say, $\sum_k y_k$). The idea was that this would still produce bounds—albeit looser ones—for individual variables but would require the solution of fewer LPs. However, we found that the looser bounds resulted in much longer time spent in the subsequent B&B calculations. So we decided to stick with calculating separate bounds for each variable.

- We use a relative optimality tolerance for fathoming. For a given tolerance ϵ, a node with lower bound v_{lb} is fathomed if $(\text{GUB} - v_{lb})/\max\{|\text{GUB}|, 1\} < \epsilon$. In our implementation, we set $\epsilon = 10^{-6}$.

- When choosing the complementarities to branch on, we employ a maximum violation approach. Given a solution $\tilde{x}$, we select the index (i, j), where $\tilde{x}_i\tilde{x}_j$ is maximum among all violated complementarities. We point out that the products $\tilde{x}_i\tilde{x}_j$ are properly scaled since every component of $\tilde{x}$ has been scaled in $[0, 1]$.

- The complementarity (8) is enforced when branching, which strengthens the relaxation further. For example, if the complementarity to be branched on involves an original variable x_j, $j \in B$, and x_j is set to be 0 in one of the children nodes to be created, then we also set $\rho_j = 0$ in that node since $1 - x_j > 0$.

All calculations have been implemented in MATLAB (version 7.8.0.347, R2009a). CPLEX is used to solve the LPs during pre-processing via the Matlab interface CPLEXINT (Baotic, 2006).

4.2 Experiments

For the computational study, we collected a total of 243 instances. All instances satisfied the assumptions of boundedness and interiority and also had $H \not\succeq 0$. These instances are of one of the four types described below:

- **BoxQP**. We selected 90 instances having $m = m_{eq} = 0$ and $(l, u) = (0, e)$. These so-called BoxQP instances were created and solved in a recent pair of papers (Burer and Vandenbussche, 2009; Vandenbussche and Nemhauser, 2005).

- **Globallib**. Globallib (Globallib) instances are standard benchmark instances in global optimization. We included 83 instances that satisfy our assumptions for (QP).

- **CUTEr**. We included 6 instances from the CUTEr test problem set (Gould et al., 2003). The CUTEr test problem set is a rich nonlinear programming problem test set. However, most QP instances in the set are convex, i.e., have $H \succeq 0$. We were only be able to find 9 instances that satisfy our selection criteria, i.e., $H \not\succeq 0$ and bounded feasible set with interior, but we excluded 3 either because of very size or numerical difficulties encountered by each of the methods.

- **RandQP**. We generated 64 instances of QPs with varying size and sparsity using a code (with slight modifications) written by Sven Leyffer for generating random mixed-integer quadratic programming instances. All instances have both inequality and equality constraints. Each element in the data matrices is a random number in a certain range. The bounds for the variables are $(l, u) = (0, e)$.

We summarize the key statistics of these instances in Table 1, including the number of instances, the number of variables, the number of constraints, and the density of H. For the sake of brevity, we only present ranges.

From now on, we refer to our method as QUADPROGBB. We will compare QUADPROGBB with the method of Burer and Vandenbussche (2008), referred to as BURVAN. Recall that both methods are based on finite branching via the KKT system, but the key differences between QUADPROGBB and BURVAN are the structure of the relaxations to be solved at each node and the algorithm

Type	# Instances	n	$m + m_{eq}$	H density
BoxQP	90	$[20, 100]$	$[0, 0]$	$[0.19, 0.99]$
Globallib	83	$[2, 100]$	$[1, 52]$	$[0.01, 1]$
CUTEr	6	$[4, 12]$	$[0, 13]$	$[0.08, 1]$
RandQP	64	$[20, 50]$	$[14, 35]$	$[0.23, 1]$

Table 1: Statistics of the test instances.

used to solve them. Specifically, the relaxations employed by BurVan include the first-order KKT conditions but do not lift the dual variables into a semidefinite matrix as a means to strengthen the relaxation and explicitly model the complementarity conditions.[2] On the other hand, the SDP relaxation utilized in Quadprogbb is derived from the completely positive representation of QP, which incorporates the dual variables and complementarity conditions in a DNN matrix. For solving the relaxations, BurVan employs an algorithm that is designed to handle Lovász-Schrijver-style SDP relaxations of quadratic programs (Burer and Vandenbussche, 2006; Lovász and Schrijver, 1991), while the algorithm we employ is specifically tailored for (DNP).

At the implementation level, another difference between Quadprogbb and BurVan is that BurVan has been designed to only solve instances of (QP) with no equality constraints, i.e., with $m_{eq} = 0$. Moreover, BurVan requires an interior point just as we do, so one cannot simply split equalities into inequalities and solve the resulting instance with BurVan. While certainly BurVan could be adapted to solve instances with $m_{eq} > 0$, we do not make such modifications to their algorithm here and only test it on instances with $m_{eq} = 0$.

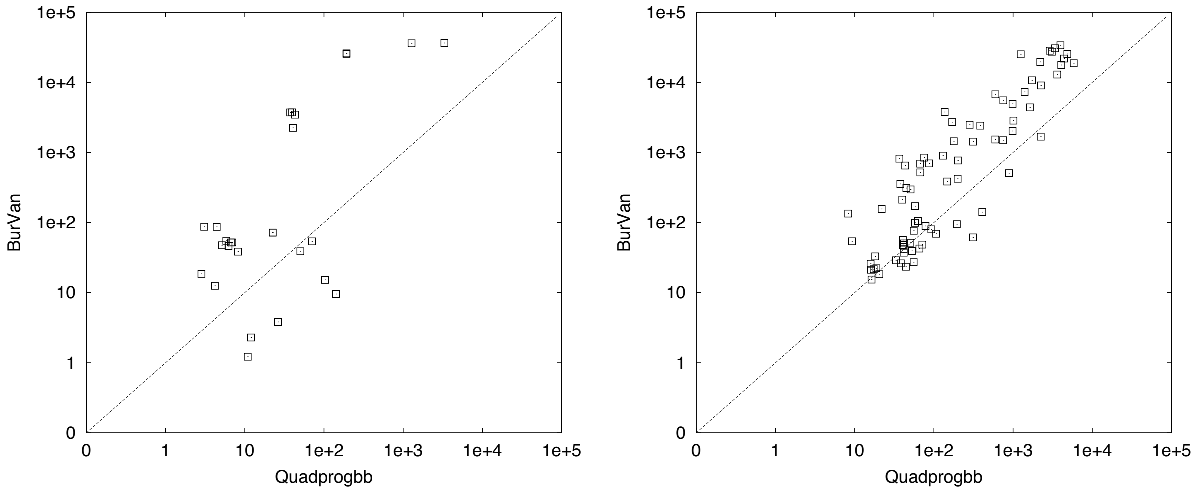
We also compare with Couenne (Belotti, 2010), a state-of-the-art open source global optimization solver. Couenne is a very different method than ours, and we summarize three major differences. First, Couenne is a general mixed-integer nonlinear solver and thus is not specialized for solving (QP). Second, Couenne utilizes a reformulation based spacial B&B technique that implements linearization, branching, heuristics to find feasible solutions and bound reduction (Belotti et al., 2009). It is well known that spatial B&B can produce an infinite number of nodes; in comparison, our approach is a finite B&B method. Third, Couenne solves LP relaxations at the nodes while we solve SDP relaxations. In particular, the computational cost at each node is relatively cheap for Couenne compared to Quadprogbb.

All instances with $m_{eq} = 0$ are solved three times, once by each of the above methods. Since BurVan cannot handle $m_{eq} > 0$, those instances with $m_{eq} > 0$ are solved by Quadprogbb and Counne only. Both Quadprogbb and BurVan use a relative tolerance of 10^{-6}. We also set Couenne's *allowable_fraction_gap* option to be 10^{-6}, which serves a similar purpose as the relative tolerance in Quadprogbb. All computations were performed on a Pentium D running at 3.2 GHz under the Linux operating system. Also a time limit of 10 hours and memory limit of 1 gigabyte were enforced for each method on each instance.

To compare Quadprogbb and BurVan, we present log-log plots of the CPU times in Figure 1, where the left plot shows Globallib and CUTEr results and the right shows BoxQP results. Each square represents one instance, and its xy-coordinates represent the CPU times of Quadprogbb and BurVan, respectively. The diagonal $y = x$ line is also plotted for reference. If a square is located above the diagonal line, Quadprogbb solves that instance faster. A similar comparison

[2] In fact, Burer and Vandenbussche (2008) do investigate relaxations that lift the dual variables into a semidefinite matrix, but the authors conclude that the increased size of the relaxations causes overall larger B&B times. So we employ their simpler relaxation in order to have the best overall times for their method.

is made between QUADPROGBB and COUENNE in Figure 2, where all four types of instances are solved. We group the results of Globallib, RandQP and CUTEr instances together and plot them in Figure 2(a) while separating the BoxQP results in Figure 2(b).



(a) Globallib and CUTEr

(b) BoxQP

Figure 1: CPU times – QUADPROGBB vs. BURVAN

Figures 1 and 2 depict only instances that were solved by both methods, i.e., instances on which neither method ran out of time or memory, and also were non-trivial. Here we regard instances requiring more than 10 seconds to solve by at least one of the three methods as "non-trivial." We summarize those instances running out of time or memory in Table 2. When a particular method ran out of resources on an instance, we calculate the relative optimality gap at the moment of termination, where the gap is defined as

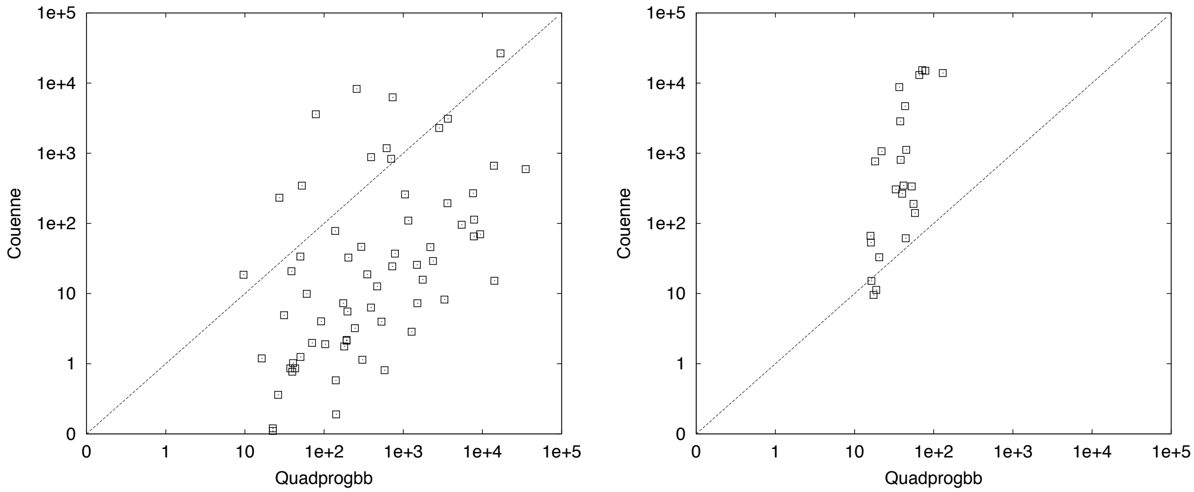
$$\frac{\text{GUB} - z_{lb}}{\max\{1, |\text{GUB}|\}} \times 100\%. \tag{20}$$

GUB denotes the current global upper bound, and z_{lb} represents the worst bound of all the remaining active nodes. In the table, we report a method's average relative optimality gap on a set of non-solved instances. We mention also that BURVAN failed numerically on one Globallib instance.

We summarize our interpretations of Figures 1 and 2 and Table 2 as follows:

- On the Globallib and CUTEr instances, QUADPROGBB performed slightly better than BURVAN. In the left plot of Figure 1, more than half of the instances are above the diagonal line. There are several squares located far above the diagonal line, indicating that QUADPROGBB performed better than BURVAN on harder instances. From Table 2, QUADPROGBB and BURVAN seem to have similar average optimality gaps on their failed Globallib instances.

- QUADPROGBB outperforms BURVAN on the BoxQP instances. This is clear among instances that were solved by both methods (see Figure 1(b)) as well as on the failed instances (see Table 2), where QUADPROGBB ran out of resources less often and, even when it did, had a better average optimality gap.



(a) Globallib, RandQP, and CUTEr

(b) BoxQP

Figure 2: CPU times – Quadprogbb vs. Couenne

- For most (but not all) Globallib, RandQP and CUTEr instances, Couenne outperforms Quadprogbb as shown in Figure 2(a). However, it is worthwhile to mention that Quadprogbb performs better than Couenne in closing the gaps on the failed instances. In particular, on the Globallib instances, the average gap when out of resources is 0.30% for Quadprogbb while it is 17.58% for Couenne; on the RandQP instances, the average gap is 4.84% for Quadprogbb while it is 42.57% for Couenne.

- For BoxQP instances, Quadprogbb significantly outperforms Couenne on instances solved by both methods (see Figure 2(b)), and Couenne runs out of memory on the larger instances leaving large optimality gaps (see Table 2). The high memory usage is due to the generation and storage of a very large B&B tree. We attribute the performance differences mainly to the relaxation differences: we use stronger SDP-based relaxations, while Couenne uses weaker LP relaxations.

To compare all three methods together and take into account failed instances, we also present their performance profiles in Figure 3. A performance profile "for a solver is the (cumulative) distribution function for ... the ratio of the computing time of the solver versus the best time of all of the solvers" over all instances (Dolan and Moré, 2002). Again we separate the instances into BoxQP instances, Figure 3(b), and all other instances, Figure 3(a). Since BurVan does not apply to RandQP instances, we does not include BurVan in Figure 3(a). We also exclude the trivial instances, i.e., instances that can be solved within 10 seconds by all the comparing methods, from the performance profiles.

The performance profiles confirmed the information presented in Figure 1 and 2 and Table 2. For Globallib, RandQP and CUTEr instances, Couenne is faster than Quadprogbb on many of them, but could not solve as many instances as Quadprogbb. On BoxQP instances, Quadprogbb is clearly the winner among all the three methods: it is not only the fastest, but also solved most of the problems.

	Type	# of Instances	# Out of Time	# Out of Memory	Avg. Rel. Opt. Gap When Out of Resources (%)
Quadprogbb	BoxQP	90	3	-	0.11
	Globallib	83	1	-	0.30
	RandQP	64	3	-	4.84
BurVan	BoxQP	90	12	-	0.87
	Globallib	69	2	-	0.17
Couenne	BoxQP	90	1	62	61.77
	Globallib	83	1	2	17.58
	RandQP	64	-	15	42.57

Table 2: Comparison of the three methods on failed instances in terms of average optimality gap. (The per-instance time limit is 10 hours, and the memory limit is 1 GB.)

5 Extension

We have implemented Quadprogbb within Matlab, and as such, it is convenient to use Quadprogbb directly or as a sub-routine to solve more complex problems. In this subsection, we provide an example to illustrate this.

Consider a problem of the form

$$\min_x \left\{ \frac{1}{2} x^T Q x + c^T x \ : \ \begin{array}{c} l \le x \le u \\ \|x\|_2 \le \Delta \end{array} \right\}, \tag{21}$$

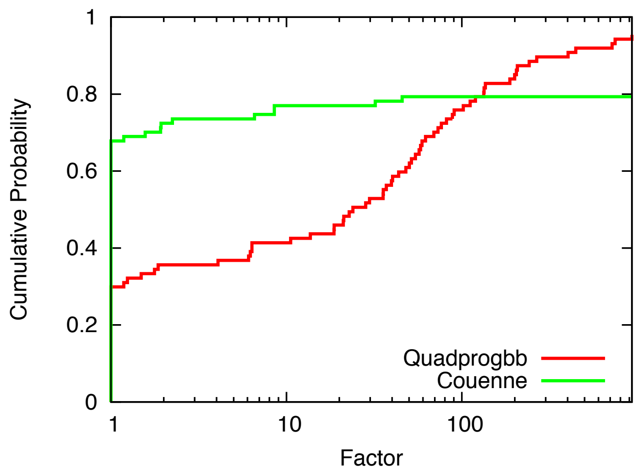
which arises, for example, as a sub-problem when applying a trust-region method to minimize a function $f(y)$ subject to bound constraints: $\min\{f(y) : \tilde{l} \le y \le \tilde{u}\}$; see Nocedal and Wright (1999) for an introduction to trust-region methods. The quadratic objective in (21) can be viewed as an approximation of $f(y)$ in trust-region specified by $\|x\|_2 \le \Delta$ around a given point $\bar{y}$, and the bounds $l \le x \le u$ reflect the bounds of the feasible region. Problem (21) would likely be solved approximately in the trust-region implementations. Nevertheless, here we are interested in solving it globally.

Using standard Lagrangian duality, it can be shown that the optimal value of (21) is equal to the optimal value of

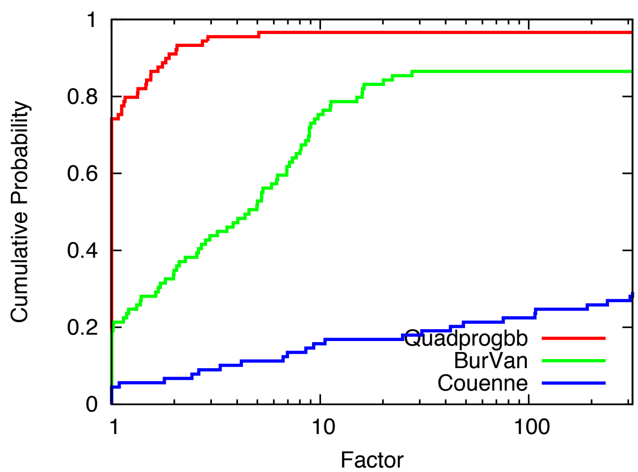
$$\max_{\lambda \ge 0} L(\lambda), \quad L(\lambda) := \min \left\{ \frac{1}{2} x^T Q x + c^T x + \lambda(x^T x - 1) \ : \ l \le x \le u \right\}, \tag{22}$$

provided that a primal-dual solution (λ^*, x^*) to (22) satisfies $\lambda^*(1 - (x^*)^T x^*) = 0$. If not, then $\lambda^*(1 - (x^*)^T x^*)$ is the associated duality gap. Note that for any given λ, $L(\lambda)$ can be evaluated by solving a box-constrained QP. In addition, $L(\lambda)$ is a concave function in λ. These facts allow us to solve (22) via bisection, for example.

We compare two ways to (approximately) solve (21): (i) via the bisection method for (22) mentioned above, where Quadprogbb is used to evaluate $L(\lambda)$; (ii) directly using Couenne. We tested the two methods on 33 randomly generated instances. In particular, we use the 33 smallest BoxQP instances ($20 \le n \le 40$) for the objective data (Q, c). Additionally, we fix $\Delta = 1$ and generate each element of l uniformly between $[-1, 0]$ and each element of u uniformly between $[0, 1]$.



(a) Globallib, RandQP, and CUTEr



(b) BoxQP

Figure 3: Performance profiles

Couenne could not solve any of these instances to within an acceptable tolerance within 1 hour, and Couenne's average relative optimality gap when out of resources was 1068%. Our method, on the other hand, solved 31 of the 33 instances within an acceptable tolerance (the average relative duality gap was 0.026%) within 1 hour. The average time for those 31 instances was 233 seconds.

As evidenced by Couenne's performance, these random instances of (21) are not "easy," and the main goal of this experiment has been to demonstrate that it is possible to solve more general problems using Quadprogbb in a convenient manner.

6 Conclusion

In this paper, we have shown how to globally solve the nonconvex quadratic programming problem (QP) by combining ideas from finite KKT-based branching and completely positive (or copositive) programming. The resultant algorithm often outperforms existing methods such as Couenne and the one by Burer and Vandenbussche (2008). In addition, the algorithm can be conveniently incorporated within other global optimization approaches.

There are several potential ways to improve the algorithm. Currently, the calculation of the bounds on the dual variables takes a considerable amount of time, especially on large instances, and each bound is calculated separately using a cold-start LP. Since the LPs for the different bounds change only in the objective, one idea would be to warm-start the next LP with the optimal basis of the current LP. Other ways to improve include a more clever choice of branching variable and enhancements to the subroutine, which solves the DNN relaxations at each node (and which we have treated as a black box in this paper).

Acknowledgments

The authors wish to thank Stefan Wild for helpful discussions, which led to the authors to the variant of the trust-region subproblem solved in Section 5.

References

M. Baotic. *CPLEXINT – Matlab interface for the CPLEX solver*, 2006. URL `http://control.ee.ethz.ch/~hybrid/cplexint.php`. Accessed January 24, 2011.

P. Belotti. Couenne: A user's manual. Technical report, Clemson University, 2010. URL `https://projects.coin-or.org/Couenne/browser/trunk/Couenne/doc/couenne-user-manual.pdf`.

P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009. URL {`http://www.optimization-online.org/DB_HTML/2008/08/2059.html`}.

S. Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495, 2009.

S. Burer. Optimizing a polyhedral-semidefinite relaxation of completely positive programs. *Mathematical Programming Computation*, 2(1):1–19, 2010.

S. Burer and D. Vandenbussche. Solving lift-and-project relaxations of binary integer programs. *SIAM J. Optim.*, 16(3):726–750 (electronic), 2006.

S. Burer and D. Vandenbussche. A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. *Math. Program.*, 113(2, Ser. A):259–282, 2008.

S. Burer and D. Vandenbussche. Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. *Computational Optimization and Applications*, 43(2):181–195, 2009.

M. R. Celis, J. E. Dennis, and R. A. Tapia. A trust region strategy for nonlinear equality constrained optimization. In *Numerical optimization, 1984 (Boulder, Colo., 1984)*, pages 71–82. SIAM, Philadelphia, PA, 1985.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002.

Globallib. Gamsworld global optimization library. `http://www.gamsworld.org/global/globallib.htm`.

N. Gould, D. Orban, and P. L. Toint. CUTEr (and SifDec): a Constrained and Unconstrained Testing Environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

N. I. M. Gould and P. L. Toint. Numerical methods for large-scale non-convex quadratic programming. In *Trends in industrial and applied mathematics (Amritsar, 2001)*, volume 72 of *Appl. Optim.*, pages 149–179. Kluwer Acad. Publ., Dordrecht, 2002.

Y. Lin and C. W. Cryer. An alternating direction implicit algorithm for the solution of linear complementarity problems arising from free boundary problems. *Appl. Math. Optim.*, 13(1): 1–17, 1985.

F. A. Lootsma and J. D. Pearson. An indefinite-quadratic-programming model for a continuous-production problem. *Philips Res. Rep.*, 25:244–254, 1970.

L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.

MathWorks. *MATLAB Reference Guide*. The MathWorks Inc., Natick, MA, 2010.

J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.*, 55(4):377–400, 1989.

T. Nguyen and R. Welsch. Outlier detection and least trimmed squares approximation using semi-definite programming. *Computational Statistics and Data Analysis*, 54:3212–3226, 2010.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

P. Pardalos. Global optimization algorithms for linearly constrained indefinite quadratic problems. *Computers and Mathematics with Applications*, 21:87–97, 1991.

P. M. Pardalos and S. A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *J. Global Optim.*, 1(1):15–22, 1991.

N. V. Sahinidis. BARON: a general purpose global optimization software package. *J. Global Optim.*, 8(2):201–205, 1996.

H. D. Sherali and W. P. Adams. *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*, volume 31 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 1999.

M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *J. ACM*, 48(2):206–242, 2001.

D. Vandenbussche and G. Nemhauser. A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Mathematical Programming*, 102(3):559–575, 2005.